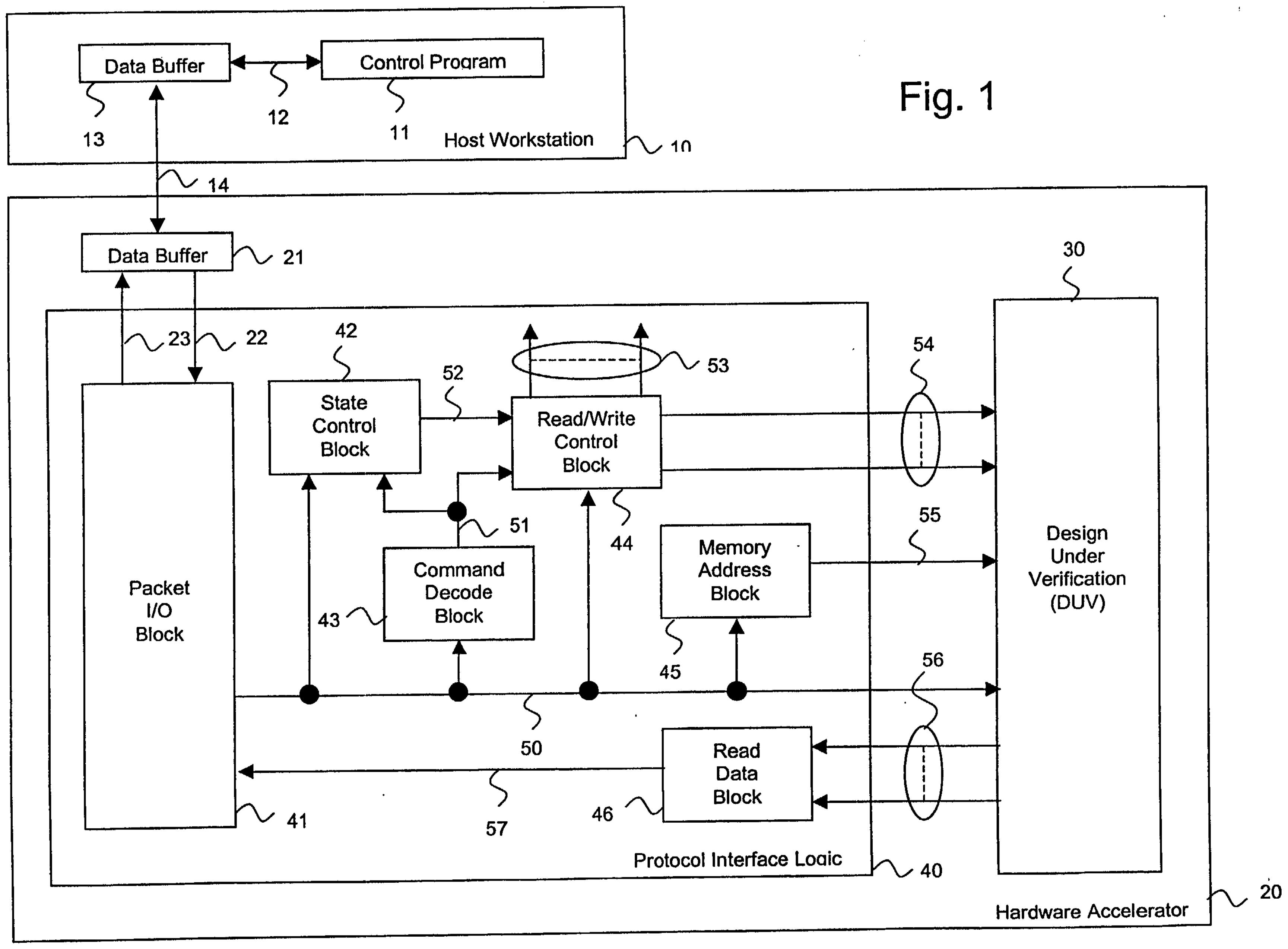
Ex. Mail No. EL659444725US
Dkt No 264/087
Sheet 1 of 18

Fig. 1



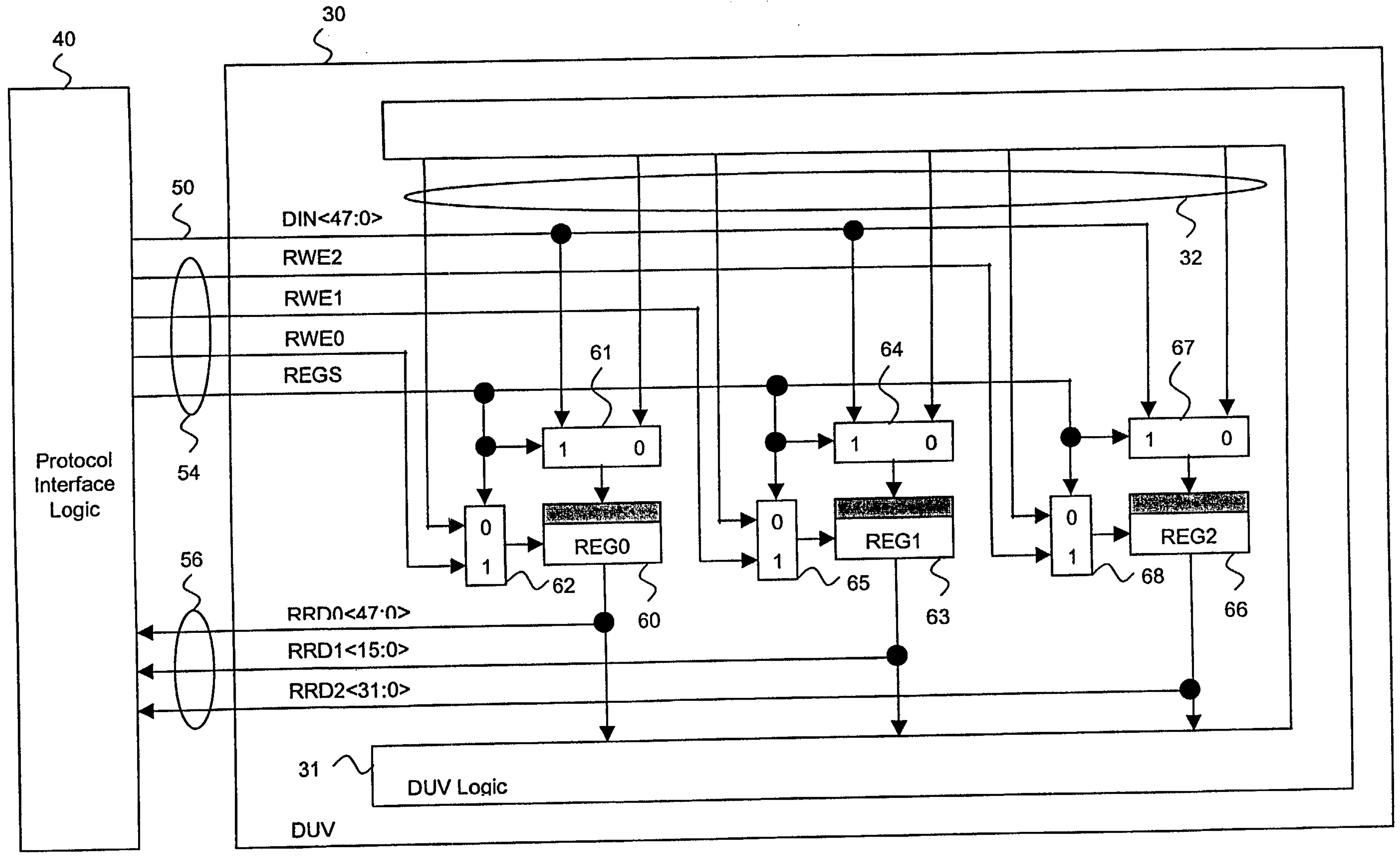

40
Protocol Interface Logic
30
50
54
56
DIN<47:0>
RWE2
RWE1
RWE0
REGS
RRD0<47:0>
RRD1<15:0>
RRD2<31:0>
32
61
64
67
1
0
REG0
REG1
REG2
62
65
68
60
63
66
31
DUV Logic
DUV

Fig. 2A

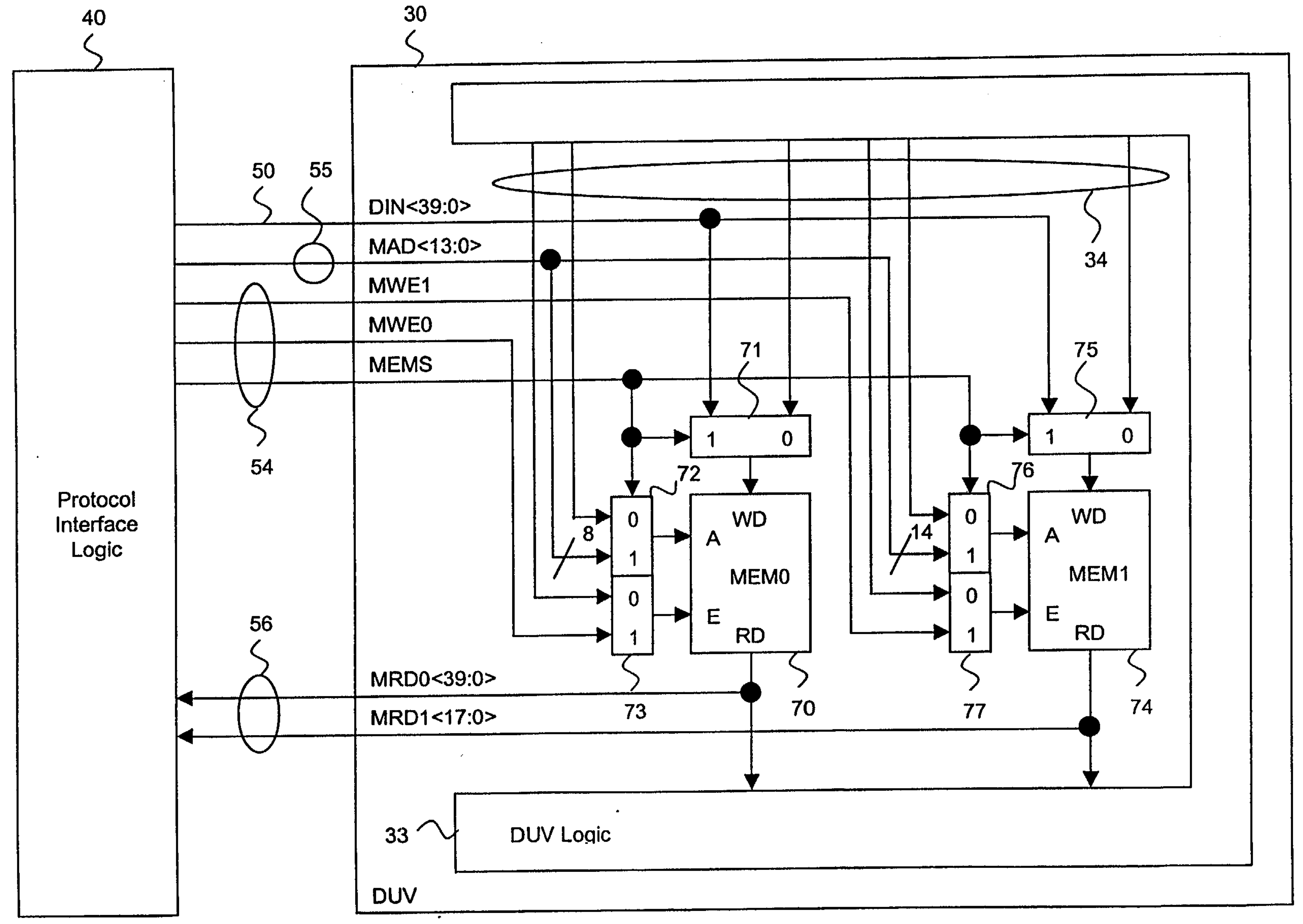

40
Protocol Interface Logic
30
50
55
DIN<39:0>
MAD<13:0>
MWE1
MWE0
MEMS
54
34
71
75
72
76
8
14
WD
A
MEM0
E
RD
MEM1
56
MRD0<39:0>
MRD1<17:0>
73
70
77
74
33
DUV Logic
DUV

Fig. 2B

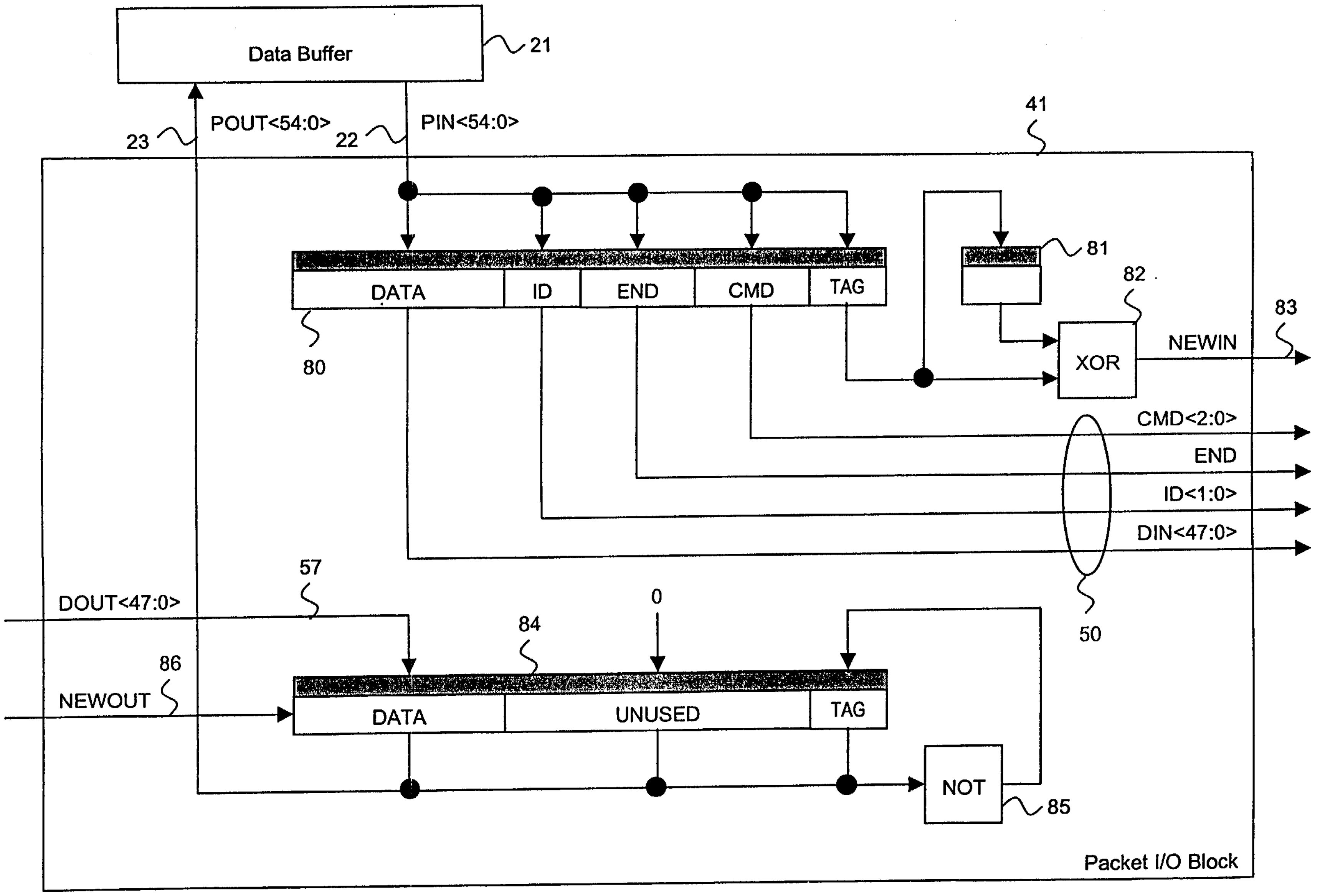

Data Buffer
21
POUT<54:0>
PIN<54:0>
23
22
41
DATA
ID
END
CMD
TAG
80
81
82
XOR
83
NEWIN
CMD<2:0>
END
ID<1:0>
DIN<47:0>
50
DOUT<47:0>
57
0
84
86
NEWOUT
DATA
UNUSED
TAG
NOT
85
Packet I/O Block

Fig. 3

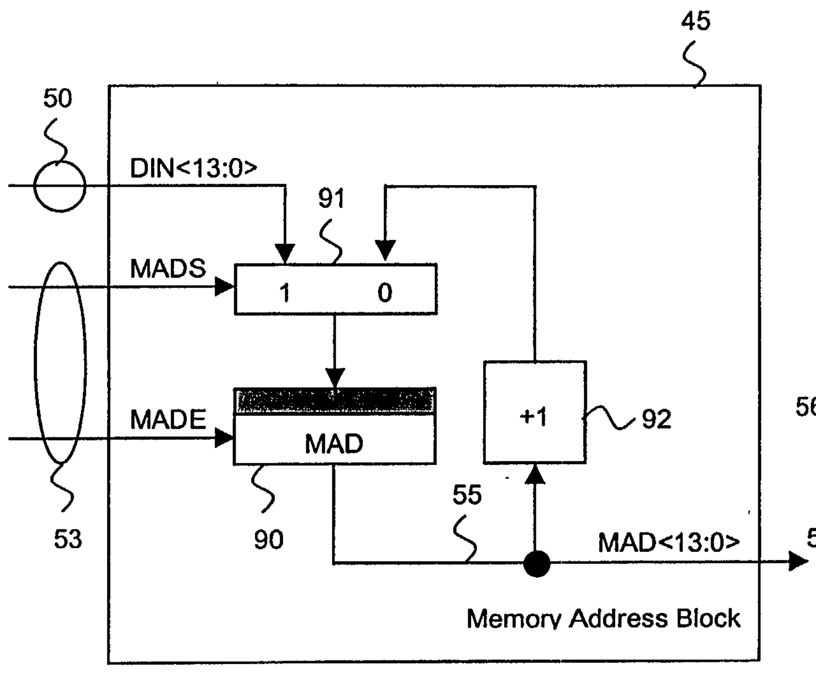

45
50
DIN<13:0>
91
MADS
1
0
MADE
MAD
+1
92
53
90
55
MAD<13:0>
Memory Address Block

Fig.4

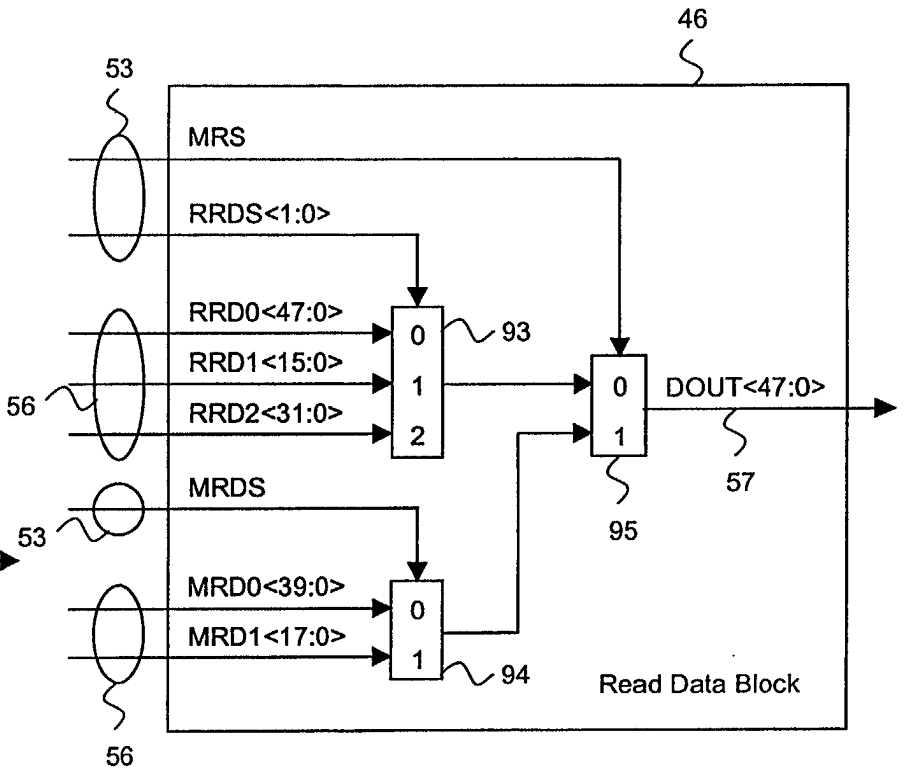

46
53
MRS
RRDS<1:0>
RRD0<47:0>
0
93
RRD1<15:0>
1
RRD2<31:0>
2
56
0
1
DOUT<47:0>
57
95
53
MRDS
MRD0<39:0>
0
MRD1<17:0>
1
94
Read Data Block
56

Fig. 5

	54 … 7	5	4	1	0
	DATA	ID	END	CMD	TAG
NOP	-	-	-	0/1	Tag
WR	Write Data	Rid	-	2	Tag
RR	-	Rid	-	3	Tag
SWM	Start Address	Mid	-	4	Tag
WM	Write Data	-	End	5	Tag
SRM	Start Address	Mid	-	6	Tag
RM	-	-	End	7	Tag

Fig. 6

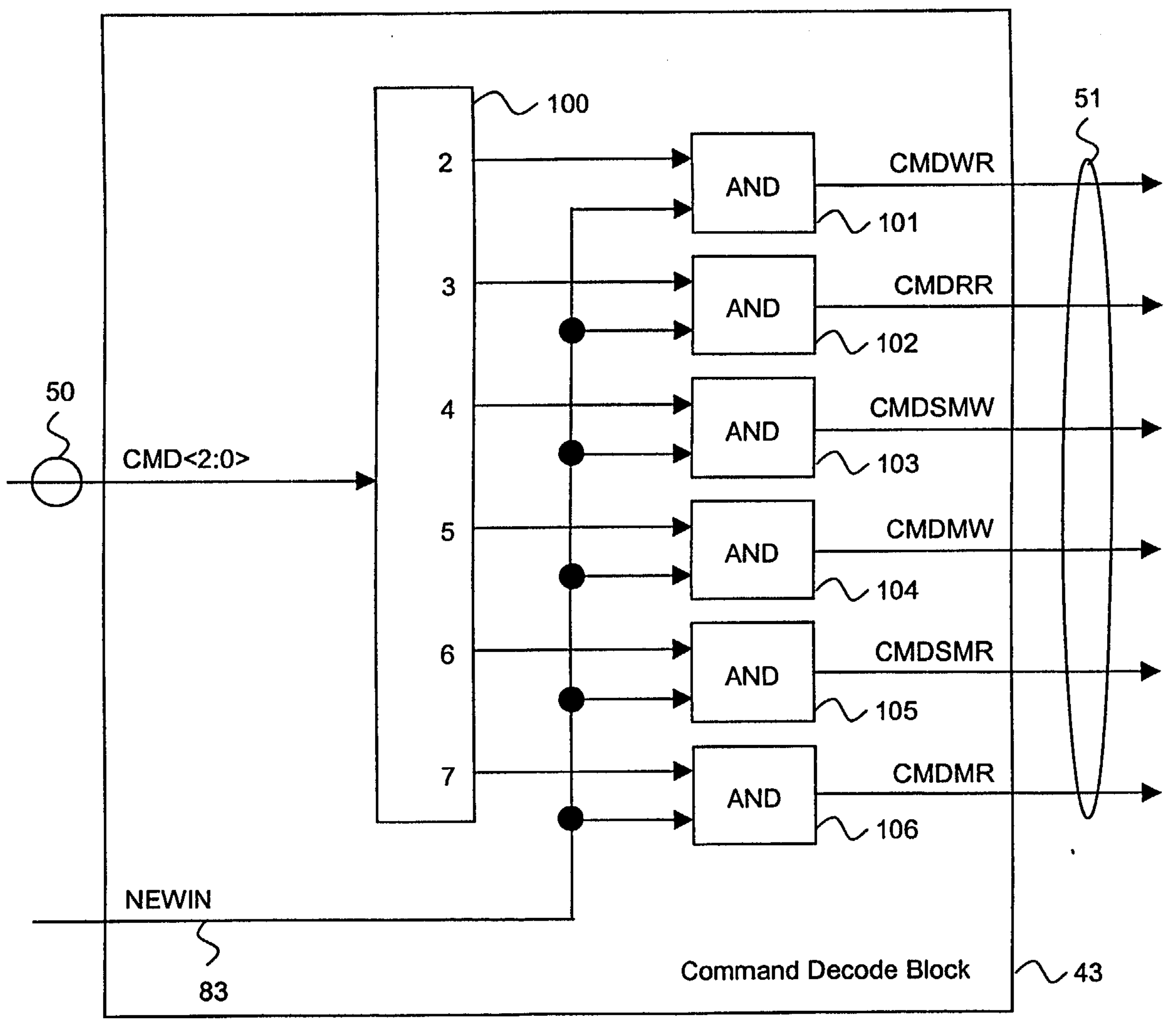

100
51
2
3
4
5
6
7
AND
CMDWR
101
CMDRR
102
CMDSMW
103
CMDMW
104
CMDSMR
105
CMDMR
106
50
CMD<2:0>
NEWIN
83
Command Decode Block
43

Fig.7

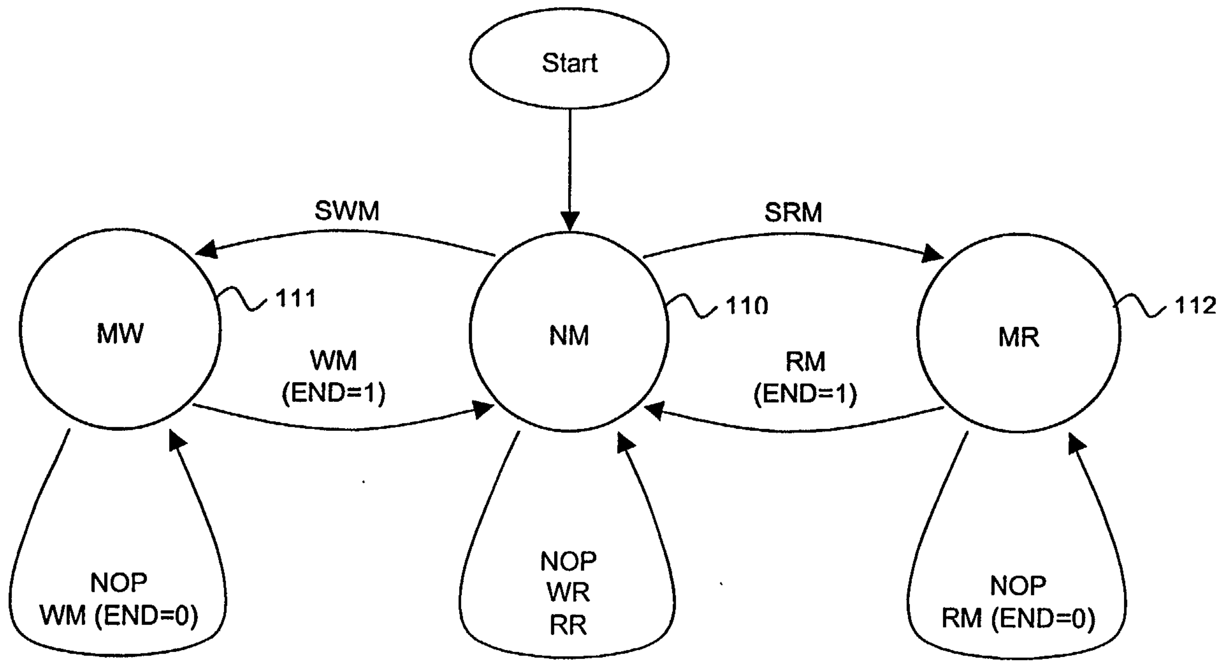

Start
SWM
SRM
MW
111
NM
110
MR
112
WM
(END=1)
RM
(END=1)
NOP
WM (END=0)
NOP
WR
RR
NOP
RM (END=0)

Fig. 8

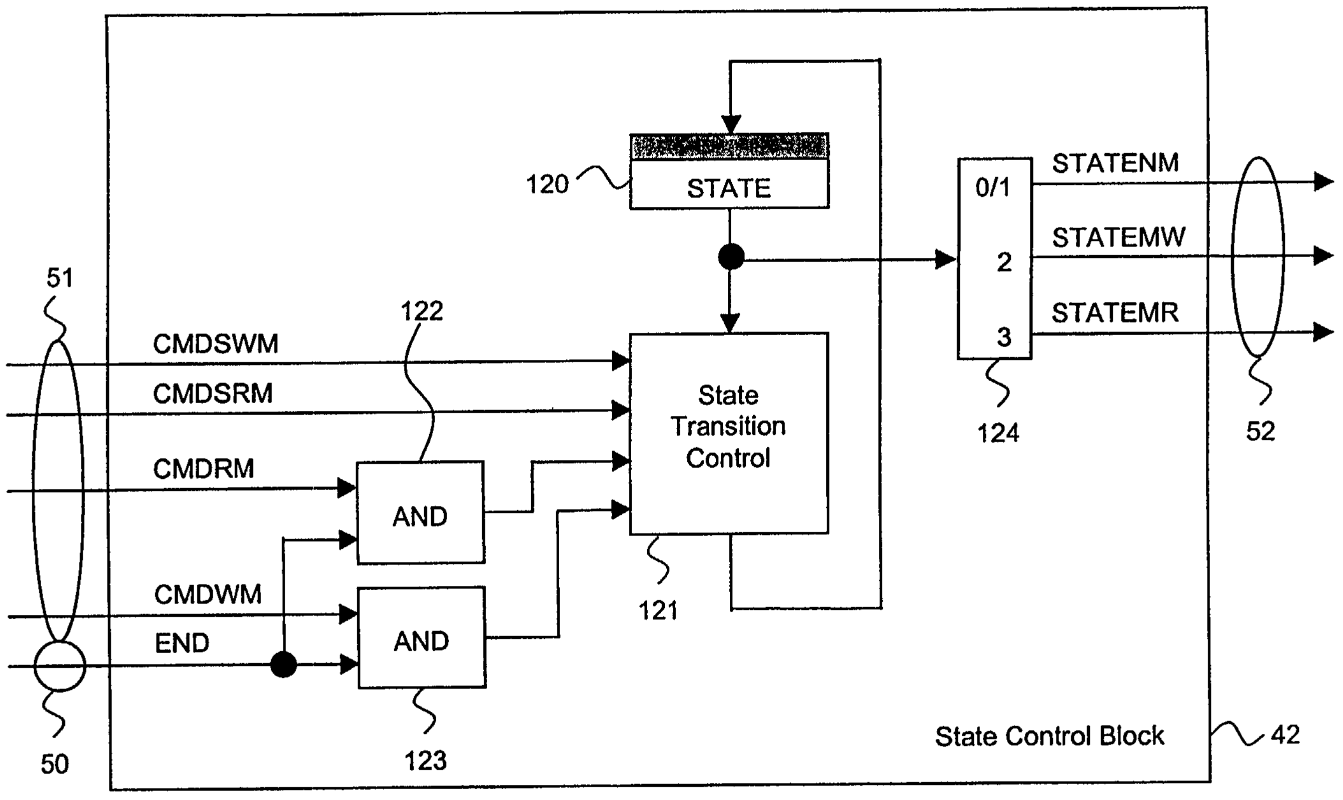

51
CMDSWM
CMDSRM
CMDRM
CMDWM
END
50
122
AND
AND
123
120
STATE
State
Transition
Control
121
0/1
2
3
124
STATENM
STATEMW
STATEMR
52
State Control Block
42

Fig. 9

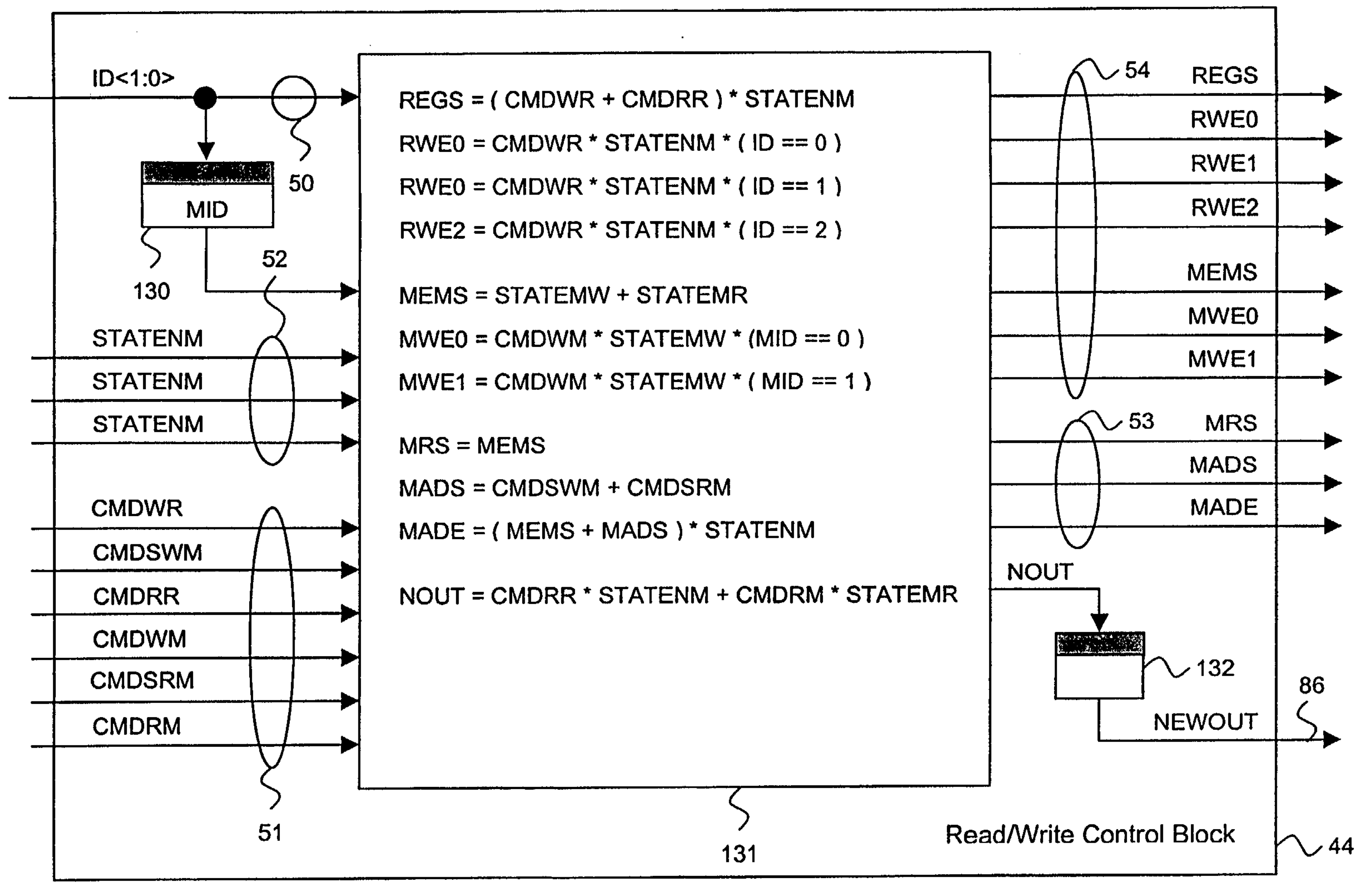

ID<1:0>
50
MID
130
52
STATENM
STATENM
STATENM
CMDWR
CMDSWM
CMDRR
CMDWM
CMDSRM
CMDRM
51
REGS = (CMDWR + CMDRR) * STATENM
RWE0 = CMDWR * STATENM * (ID == 0)
RWE0 = CMDWR * STATENM * (ID == 1)
RWE2 = CMDWR * STATENM * (ID == 2)
MEMS = STATEMW + STATEMR
MWE0 = CMDWM * STATEMW * (MID == 0)
MWE1 = CMDWM * STATEMW * (MID == 1)
MRS = MEMS
MADS = CMDSWM + CMDSRM
MADE = (MEMS + MADS) * STATENM
NOUT = CMDRR * STATENM + CMDRM * STATEMR
131
54
REGS
RWE0
RWE1
RWE2
MEMS
MWE0
MWE1
53
MRS
MADS
MADE
NOUT
132
86
NEWOUT
Read/Write Control Block
44

Fig.10

140	CYCLE	N	N+1	N+2	N+3	N+4	N+5
141	NEWIN	1	1	0	1	1	0
142	CMD	WR	WR	-	WR	WR	-
143	DIN	A	B	-	C	D	-
144	REG	-	A	B	-	C	D

Fig.11A

140	CYCLE	N	N+1	N+2	N+3	N+4	N+5
141	NEWIN	1	-	-	1	-	-
142	CMD	RR	-	-	RR	-	-
144	REG	A	-	-	B	-	-
145	POUT(DATA)	-	A	A	A	B	B
146	TRANSFER	-	-	A	-	-	B

Fig.11B

140	CYCLE	N	N+1	N+2	N+3	N+4	N+5
141	NEWIN	1	1	1	0	1	0
142	CMD	SWM	WM	WM	-	WM	-
143	DIN	10	A	B	-	C	-
147	MAD	-	10	11	12	12	13
148	MEM	-	A	B	-	C	-

Fig.11C

140	CYCLE	N	N+1	N+2	N+3	N+4	N+5	N+6
141	NEWIN	1	1	0	0	1	0	0
142	CMD	SRM	RM	-	-	RM	-	-
143	DIN	10	-	-	-	-	-	-
147	MAD	-	10	11	11	11	12	11
148	MEM	-	A	-	-	B	-	-
145	POUT(DATA)	-	-	A	A	A	B	B
146	TRANSFER	-	-	-	A	-	-	B

Fig.11D

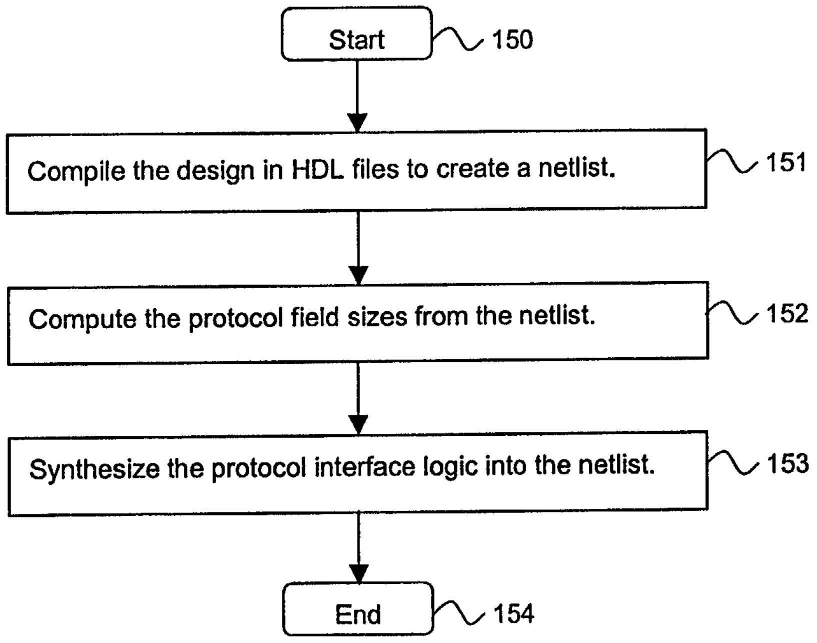

Start
150
Compile the design in HDL files to create a netlist.
151
Compute the protocol field sizes from the netlist.
152
Synthesize the protocol interface logic into the netlist.
153
End
154

Fig.12

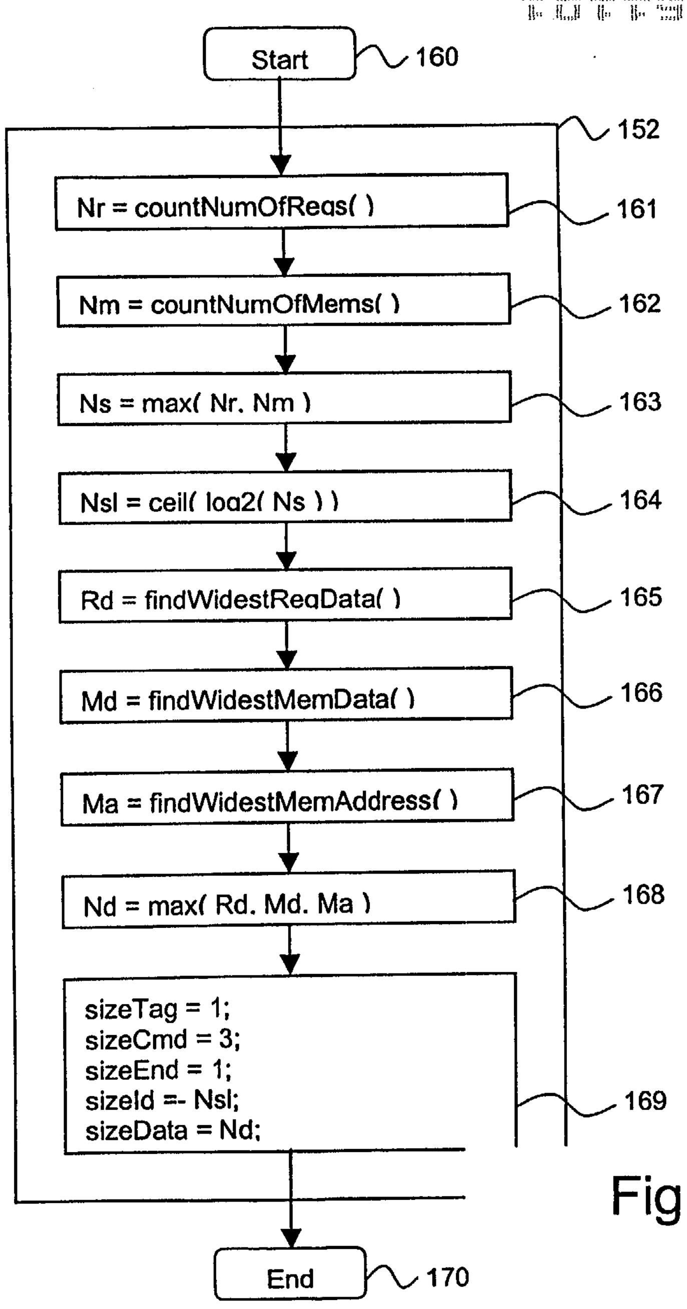

Start
160
152
Nr = countNumOfReqs( )
161
Nm = countNumOfMems( )
162
Ns = max(Nr. Nm)
163
Nsl = ceil(log2(Ns))
164
Rd = findWidestReqData( )
165
Md = findWidestMemData( )
166
Ma = findWidestMemAddress( )
167
Nd = max(Rd. Md. Ma)
168
sizeTag = 1;
sizeCmd = 3;
sizeEnd = 1;
sizeId =- Nsl;
sizeData = Nd;
169
End
170

Fig. 13

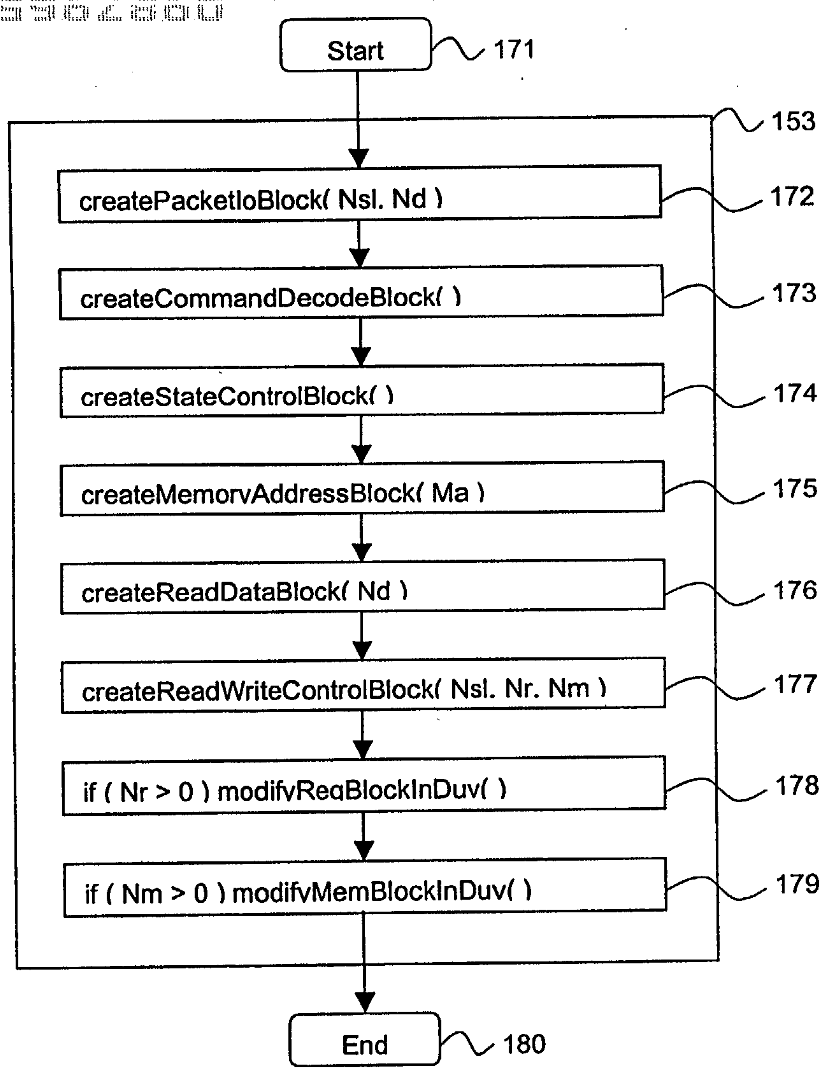
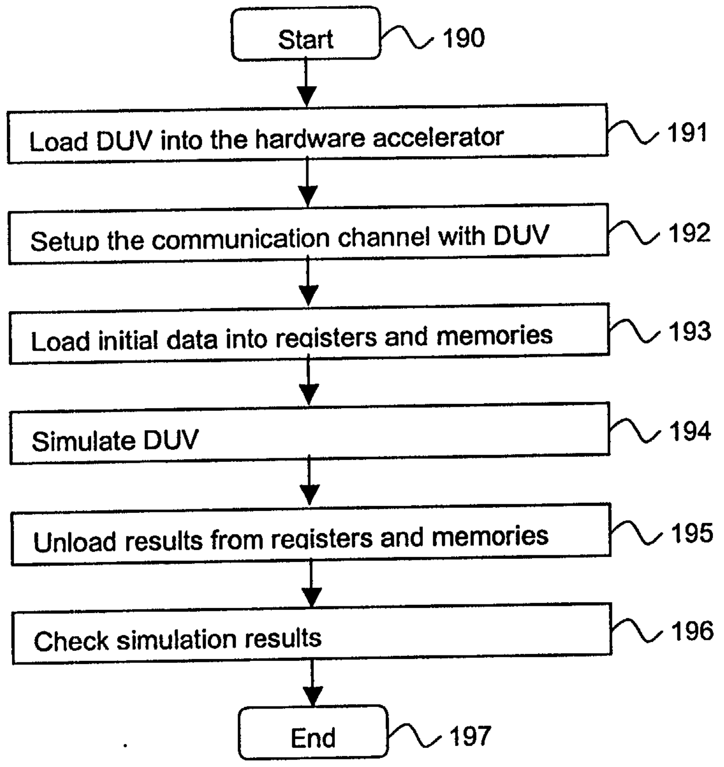

Start
171
153
createPacketIoBlock(Nsl. Nd)
172
createCommandDecodeBlock( )
173
createStateControlBlock( )
174
createMemoryAddressBlock(Ma)
175
createReadDataBlock(Nd)
176
createReadWriteControlBlock(Nsl. Nr. Nm)
177
if (Nr > 0) modifyReqBlockInDuv( )
178
if (Nm > 0) modifyMemBlockInDuv( )
179
End
180

Fig.14

[illegible]

Ex. Mail No. EL659444725US
Dkt. No 264/087
Sheet 15 of 18



Start
190
Load DUV into the hardware accelerator
191
Setup the communication channel with DUV
192
Load initial data into registers and memories
193
Simulate DUV
194
Unload results from registers and memories
195
Check simulation results
196
End
197

Fig.15

```
#define CMD_NOP 0  /* no operation        */
#define CMD_WR  2  /* write register      */
#define CMD_RR  3  /* read register       */
#define CMD_SWM 4  /* select write memory */
#define CMD_WM  5  /* write memory        */
#define CMD_SRM 6  /* select read memory  */
#define CMD_RM  7  /* read memory         */

#define uint unsigned int

int sizeId;   /* size of ID field in bits */
int sizeData; /* size of DATA field in bits */

uint tagOut = 0;  /* outgoing packet tag */
uint tagIn = 0;   /* incoming packet tag */

uint * packetOut;  /* outgoing packet */
uint * packetIn;   /* incoming packet */

extern void put_bits( uint *packet, int pos, int size, uint data );
extern void get_bits( uint *packet, int pos, int size, uint *data );
extern void copy_to_dbuff( int size, uint *packet );
extern void copy_from_dbuff( int size, uint *packet );
extern void transfer_to_haccel( int size );
extern void transfer_from_haccel( int size );

```

Fig.16

```
void sendPacket( uint cmd, uint end, uint id, int ds, uint data )
{
    ps = 5 + sizeId + ds;
    tagOut = tagOut ^ 1;
    put_bits( packetOut, 0, 1, tagOut );
    put_bits( packetOut, 1, 3, cmd );
    put_bits( packetOut, 4, 1, end );
    put_bits( packetOut, 5, sizeId, id );
    put_bits( packetOut, 5+sizeId, ds, data );
    copy_to_dbuff( ps, packetOut );
    transfer_to_haccel( ps );
}

void receivePacket( int ds, uint *data )
{
    ps = 5 + sizeId + ds;
    while ( 1 ) {
        transfer_from_haccel( ps );
        copy_from_dbuff( ps, packetIn );
        get_bits( packetIn, 0, 1, &tag );
        if ( tag != tagIn ) break;
    }
    tagIn = tag;
    get_bits( packetIn, 5+sizeId, ds, data );
}
```

Fig.17

```
void writeReg( uint rid, int ds, uint data )
{
    sendPacket( CMD_WR, 0, rid, ds, data );
}

void readReg( uint rid, int ds, uint data )
{
    sendPacket( CMD_RR, 0, rid, 0, NULL );
    receivePacket( ds, &data );
}

void writeMem( uint mid, int as, uint start,
               int nw, int ds, uint *data )
{
    sendPacket( CMD_SWM, 0, mid, as, start );
    end = 0;
    for ( i = 0; i < nw; i++ ) {
        if ( i == (nw-1) ) end = 1;
        sendPacket( CMD_WM, 0, end, ds, data[i] );
    }
}

void readMem( uint mid, int as, uint start,
              int nw, int ds, uint *data )
{
    sendPacket( CMD_SRM, 0, mid, as, start );
    end = 0;
    for ( i = 0; i < nw; i++ ) {
        if ( i == (nw-1) ) end = 1;
        sendPacket( CMD_RM, 0, end, ds, data[i] );
        receivePacket( ds, &data[i] );
    }
}
```

Fig.18